(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 481 231 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
26.11.1997 Bulletin 1997/48

(51) Int Cl.^{6}: **G06F 11/00**

(21) Application number: **91115808.7**

(22) Date of filing: **18.09.1991**

(54) A method and system for increasing the operational availability of a system of computer programs operating in a distributed system of computers

Verfahren und System zur Erhöhung der Betriebsverfügbarkeit eines Systems von Rechnerprogrammen, wirkend in einem verteilten Rechnerssystem

Procédé et système d'augmentation de la disponibilité opérationnelle d'un système de programmes d'ordinateur opérant dans un système distribué d'ordinateurs

(84) Designated Contracting States:
DE FR GB

(30) Priority: **17.10.1990 US 599178**

(43) Date of publication of application:
22.04.1992 Bulletin 1992/17

(73) Proprietor: **International Business Machines Corporation**
Armonk, N.Y. 10504 (US)

(72) Inventor: **Smith, Donald M.**
Germantown, Maryland 20874 (US)

(74) Representative: **Rach, Werner, Dr. et al**
IBM Deutschland Informationssysteme GmbH,
Patentwesen und Urheberrecht
70548 Stuttgart (DE)

(56) References cited:

- **PROCEEDINGS OF THE 9TH SYMPOSIUM ON RELIA- BLE DISTRIBUTED SYSTEMS, October 9- 11, 1990, pages 2-11, IEEE, New York, US; T.P. NG:'The design and implementation of a Reliable Distributed Operating System ROSE'**
- **PROCEEDINGS OF THE 6TH SYMPOSIUM ON RELIA- BILITY IN DISTRIBUTED SOFTWARE &**
- **DATABASE SYSTEMS, March 17-19, 1987, pages115-125, IEEE, New York, US; M. AHAMAD et al.: Fault tolerant computing in object based distributed operating systems'**
- **PROCEEDINGS OF THE 1986 INTERNATIONAL CON- FERENCE ON COMPUTER LANGUAGES, October 27 30, 1986, pages 98-106, IEEE, New York, US K.-J. LIN:'Resilient procedures an approach to highly available system'**

Note: Within nine months from the publication of the mention of the grant of the European patent, any person may give notice to the European Patent Office of opposition to the European patent granted. Notice of opposition shall be filed in a written reasoned statement. It shall not be deemed to have been filed until the opposition fee has been paid. (Art. 99(1) European Patent Convention).

Printed by Jouve, 75001 PARIS (FR)

Description

The invention disclosed broadly relates to data processing systems and more particularly relates to systems and methods for enhancing fault tolerance in data processing systems.

Operational availability is defined as follows: "If a stimulus to the system is processed by the system and the system produces a correct result within an allocated response time for that stimulus, and if this is true for all stimuli, then the system's availability is 1.0."

It is recognized that there are many contributors to high operational availability: (1) failures in both the hardware system and the software system must be detected with sufficiently high coverage to meet requirements; (2) the inherent availability of the hardware (in terms of simple numerical availability of its redundancy network), including internal and external redundancy must be higher than the system's required operational availability; and (3) failures in the software must not be visible to or adversely affect operational use of the system. This invention addresses the third of these contributors with the important assumption that software failures due to design errors and to hardware failures, will be frequent and hideous.

The prior art has attempted to solve this type of problem by providing duplicate copies of the entire software component on two or more individual processors, and providing a means for communicating the health of the active processor to the standby processor. When the standby processor determines, through monitoring the health of the active processor that the standby processor must take over operations, the standby processor initializes the entire software component stored therein and the active processor effectively terminates its operations. The problem with this approach is that the entire system is involved in each recovery action. As a result, recovery times tend to be long, and failures in the recovery process normally render the system inoperable. In addition, if the redundant copies of the software systems are both normally operating (one as a shadow to the other), then the effect of common-mode failures is extreme and also affects the whole system.

An approach solving the above problem is disclosed in the Proceedings of the Ninth Symposium on Reliable Distributed Systems, 9-11 October 1990, Alabama, entitled 'The Design and Implementation of a Reliable Distributed Operating System - ROSE' by T. P. NG. Therein proposed is a modular distributed operating system that provides support for building reliable applications and handling hardware failures. The aim is to increase data availability despite those failures. Provided are so-called Replicated Address Space (RAS) objects whose content is accessible with a high probability. Further, a so-called Resilient Process (RP) abstraction allows user processes to survive hardware failures with minimal interruption. In particular two different implementations of the Resilient Processes are discussed, one who checkpoints the information about its state in an RAS object periodically, the other which uses replicated execution by executing the same code in different nodes at the same time. 'Modularity' of the distributed operating system in this sense means that the kernel layer of the operating system supports multiple processes with separate address spaces. Within each process and its address space, multiple tasks can execute concurrently and communicate with one another by a shared memory. Several processes can replicate and share a portion of their address space. Several processes replicate the execution of an application to provide the image of a single but very reliable process. The application may set up two processes that belong to the same configuration object, and share an RAS object. In case of a failure of one of the processes the second process would reconfigure to remove the first process from the configuration. If the reconfiguration is successful, the second process would read the content of the RAS object and resume processing were the first process has left off.

It is therefore an object of the invention to increase the operational availability of a system of computer programs operating in a distributed system of computers where recovery from a failure of software or hardware occurs before the failure becomes operationally visible.

It is another object of the invention to provide fault tolerance in a system of computer programs operating in a distributed system of computers, having a high availability and fast recovery time.

It is still a further object of the invention to provide improved operational availability in a system of computer programs operating in a distributed system of computers, with less software complexity, than was required in the prior art.

These objects, features and advantages of the invention are by the method as set forth in claim 1. This invention provides a mechanism to organize the computer software in such a way that its recovery from a failure (of either itself or the hardware) occurs before the failure becomes operationally visible. In other words, the software is made to recover from the failure and reprocess or reject the stimulus so that the result is available to the user of the system within the specified response time for that type of stimulus.

A software structure that will be referred to as an operational unit (OU), and a related availability management function (AMF) are the key components of the invention. The OU and portions of the AMF are now described. The OU concept is implemented by partitioning as much of the system's software as possible into independent self-contained modules whose interactions with one another is via a network server. A stimulus enters the system and is routed to the first module in its thread, and from there traverses all required modules until an appropriate response is produced and made available to the system's user.

Each module is in fact two copies of the code and data-space of the OU. One of the copies, called the Primary Address Space (PAS), maintains actual state data. The other copy, called the Standby Address Space (SAS), runs in a separate processor, and may, or may not maintain actual state data, as described later.

The Availability Management Function (AMF) controls the allocation of PAS and SAS components to the processors. When the AMF detects an error, a SAS becomes PAS and the original PAS is terminated. Data servers in the network are also informed of the change so that all communication will be redirected to the new PAS. In this fashion, system availability can be maintained.

These and other objects, features and advantages of the invention will be more fully appreciated with reference to the accompanying figures.

Fig. 1 is a timing diagram illustrating response time allocation.

Fig. 2A is a schematic diagram of several operational units in a network.

Fig. 2B is an alternate diagram of the operational unit architecture illustrating the use of a data server OU by several applications.

Fig. 2C is an illustration of the operational unit architecture.

Fig. 2D is an illustration of the operational unit architecture showing an example of the allocation of general operational units across three groups.

Fig. 3A-F shows of the operational unit at various stages during the initialization and operation, reconfiguration and recovery functions.

Fig. 1 shows a timing diagram illustrating response time allocation. The required response time (Tmax) between a stimulus input and its required output suballocated so that a portion of it is available for normal production of the response (Tnormal), a portion is available for unpredicted resource contention (Tcontention), and a portion is available for recovery from failures (Trecovery). The first of these, Tnormal, is divided among the software and hardware elements of the system in accordance with their processing requirements. This allocation will determine the performance needed by each hardware and software component of the system. The second portion, Tcontention, is never allocated. The last portion, Trecovery, is made sufficiently long that it includes time for error detection (including omission failures), hardware reconfiguration, software reconfiguration, and reproduction of required response. A rule of thumb is to divide the required response time Tmax in half and subdivide the first half so that one quarter of the required response time is available for normal response production Tnormal, and the second quarter is available for unpredicted resource contention, Tcontention. The second half of the response time, Trecovery, is then available for failure detection and response reproduction.

The specific problem addressed by this invention is how to reduce the time required for hardware and software reconfiguration, Trecovery, of a complex system to a small fraction of Tmax. This problem is solved by a software structure that will be referred to as an operational unit (OU), and a related availability management function (AMF). The OU and portions of the AMF are now described.

Referring to Fig. 2A, the OU concept is implemented by partitioning as much of the system's software as possible into independent self-contained modules or OUs 10 whose interactions with one another are via a network server 12. None of these modules shares data files with any other module, and none of them assumes that any other is (or is not) in the same machine as itself. A stimulus 14 enters the system and is routed to the first module in its thread and from there traverses all requested modules until an appropriate response is produced and made available to the system's user 16.

Each module maintains all necessary state data for its own operation. If two or more modules require access to the same state knowledge then: (1) each must maintain the knowledge; (2) updates to that knowledge must be transmitted among them as normal processing transactions; or (3) each must be tolerant of possible difference between its knowledge of the state, and the knowledge of the others. This tolerance may take on several forms depending on the application, and may include mechanisms for detecting and compensating (or correcting) for differences in state. If two modules 10 absolutely must share a common data base for performance or other reason, then they are not "independent" and are combined into a single module for the purpose of this invention.

It is acceptable for one module 10' to perform data server functions for multiple other modules (as shown in Fig. 2B) providing those other modules 10 can operationally compensate for failure and loss of the server function. Compensate means that they continue to provide their essential services to their clients, and that the inability to access the common state does not result in unacceptable queuing or interruption of service. Clearly this constrains the possible uses of such common servers.

Finally, a module 10 must provide predefined degraded or alternative modes of operation for any case where another module's services are used, but where that other module is known to be indefinitely unavailable. An exception to this rule is that if a server is part of a processor (if it deals with the allocation or use of processor resources), then the module may have unconditional dependencies on the server. In this case, failure of the server is treated by the availability management function as failure of the processor. If a module conforms to all of the above conditions, then it becomes an OU through the application of the following novel structuring of its data, its logic and its role within the system. This structure is shown in Fig. 2C.

Two complete copies of the modules are loaded into independent address spaces 20, 20' in two separate computers 22, 22'. One of these modules is known by the network server as the Primary Address Space (PAS) and is the one to which all other module's service requests are directed. The other of these modules is called the Standby Address Space (SAS), and is known only by the PAS. It is invisible to all other modules. The PAS sends application dependent state data to the SAS so that the SAS is aware of the state of the PAS. Whether the interface between the PAS and the SAS of an OU is a synchronous commit or is an unsynchronized interface is not limited by this invention, and is a trade-off between the steady-state response time, and the time required for a newly promoted PAS to synchronize with its servers and clients. This trade-off is discussed in strategy #1 below.

The PAS maintains the state necessary for normal application processing by the module. The SAS maintains sufficient state knowledge so that it can transition itself to become the PAS when and if the need should arise. The amount of knowledge this requires is application dependent, and is beyond the scope of this invention.

Both the PAS and the SAS of an OU maintain open sessions with all servers of the OU. The SAS sessions are unused until/unless the SAS is promoted to PAS. When the SAS is directed by the AMF to assume the role of PAS, it assures that its current state is self-consistent (a failure may have resulted from the PAS sending only part of a related series of update messages), and then communicates with clients and servers of the PAS to establish state agreement with them. This may result in advancing the SAS's state knowledge or in rolling back the state knowledge of the clients and servers. Any rolling back must be recovered by reprocessing any affected stimuli, and/or by informing users that the stimuli have been ignored. Simultaneous with this process, the network server is updated so that it directs all new or queued service requests to the SAS instead of the PAS. This last action constitutes promotion of the SAS to the position of PAS, and is followed by starting up a new SAS in a processor other than the one occupied by the new PAS.

Several strategies for maintenance of standby data by the PAS are relevant to the invention. They are summarized as follows.

Strategy #1. The SAS may retain a complete copy of the state of the PAS. If the copy is committed to the SAS before the PAS responds to its stimulus, then the restart recovery will be very fast, but the response time for all stimuli will be the longest. This approach is superior if response time requirements allow it.

Strategy #2. The SAS may retain a "trailing" copy of the state of the PAS. Here, the PAS sends state updates as they occur, at the end of processing for a stimulus, or batches them for several stimuli. The SAS trails the PAS state in time and must therefore be concerned with state consistency within its data and between itself and its servers and clients. This yields very fast steady-state response time but requires moderate processing during failure recovery.

Strategy #3. The SAS may retain knowledge of the stimuli currently in process by the PAS so that at failure, the exact state of the relationships between the PAS and its clients and servers is known by the SAS. This requires commitment of beginning and end of transactions between the PAS and the SAS, but reduces the inter-OU synchronization required during failure recovery.

These mechanisms may be used alone or in various combinations by an OU. The determination of which to use is a function of the kinds of stimuli, the response time requirements, and the nature of the state retained by the application.

The Availability Management Function (AMF)

The characteristics just described comprise an OU, but do not by themselves achieve availability goals. Availability goals are achieved by combining this OU architecture with an AMF that controls the state of all OU's in the system. The AMF has three components, each with its own roles in the maintenance of high availability of the system's OU's. The relationship between an OU and the AMF is illustrated in Figs. 3A through 3F and is described below.

The most important AMF function is that of group manager. A group is a collection of similarly configured processors that have been placed into the network for the purpose of housing a predesignated set of one or more OU's. Each group in the system (if there are more than one) is managed independently of all other groups. In Fig. 2D, three groups are shown. Here, the OU's residing in each group (rather than the processors) are shown.

The number of processors in each group, and the allocation of OU PAS and SAS components to those processors may be widely varied in accordance with the availability requirement, the power of each processor, and the processing needs of each PAS and SAS component. The only constraint imposed by the invention is that the PAS and the SAS of any single OU must be in two different processors if processor failures are to be guarded against.

Group Management: Referring to Fig. 3A, the group manager resides in one of the processors of a group (the processor is determined by a protocol not included in this invention). It initializes the group for cold starts, and reconfigures the group when necessary for failure recovery. Within the group, each OU's PAS and SAS exist on separate processors, and sufficient processors exist to meet inherent availability requirements. The group manager monitors performance of the group as a unit and coordinates the detection and recovery of all failures within the group that do not require attention at a system level. This coordination includes the following:

1. Commanding the takeover of an OU's functional responsibilities by the SAS when it has been determined that a failure has occurred in either the PAS, or in the processor or related resource necessary to the operation of the PAS.

2. Initiating the start-up of a new address space to house a new SAS after a prior SAS has been promoted to PAS.

3. Updating the network server's image of the location of each OU as its PAS moves among the processors of a group in response to failures or scheduled shutdown of individual group processors.

Errors detected by the group manager include processor failure (by heartbeat protocols between group members), and failures of an OU that affect both its PAS and SAS, for example, those caused by design errors in the communication between them or the common processing of standby/backup state data. The mechanisms for error detection are beyond the scope of the invention.

Local Management: Group level support of the OU architecture relies on the presence of certain functions within each processor of the group. These functions are referred to as the AMF's local manager 32. The local manager is implemented within each processor as an extension of the processor's control program and is responsible for detecting and correcting failures that can be handled without intervention from a higher (group or above) level. The local manager maintains heartbeat protocol communications with each OU PAS or SAS in its processor to watch for abnormalities in their performance. It also receives notification from the operating system of any detected machine level hardware or software problem. Any problem that cannot be handled locally is forwarded to the group manager for resolution.

Global Management: The isolation and correction of system level problems and problems associated with the network fall with the AMF's global manager 34. The global manager 34 correlates failures and states at the lower levels to detect and isolate errors in network behavior, group behavior, and response times of threads involving multiple processors. It also interacts with human operators of the system to deal with failures that the automation cannot handle. The global manager is designed to operate at any station in the network, and is itself an OU. The movement of the global manager 34 OU from one group to another, if necessary, is initiated and monitored by the human operator using capabilities contained in the local manager at each processor station.

Network Management: The network manager is a part of the AMF global manager 34. Its functionality and design is configuration dependent, and is beyond the scope of this invention.

Figs. 3A-F show the functionality for the AMF during initialization, operation, reconfiguration and recovery of the system. In Fig. 3A, PAS and SAS are loaded and initialized. OU locations are entered in the network server. Fig. 3B shows the synchronization of the PAS and SAS with clients and servers to establish consistency of states. Fig. 3C shows steady state operation with PAS responding to a stimulus and outputting a response. The SAS is kept updated by the PAS.

Fig. 3D shows what happens when the PAS fails. The old SAS is promoted to new PAS and a new SAS is loaded and initialized. The network servers are also updated with the new location of the OU. Fig. 3E shows re-synchronization of the new PAS with the clients and servers. At steady state (Fig. 3E), the new PAS is responding to stimuli as normal.

It is the combination of the OU structure and the recovery functions of the AMF that constitute this invention. By comparison, the contemporary strategy and mechanisms of software failover/switchover deal with units of entire processors rather than with smaller units of software as in this invention. Because the unit of failure and recovery achieved by this invention is small by comparison, the time for recovery is also small. Furthermore, the recovery from processor level failures can be accomplished in small steps spread across several processors (as many as are used to contain SAS's for the PAS's contained in the failed processor). This is crucial to maintaining continuous high availability operation in large real time systems.

Claims

1. A method for increasing the operational availability of a system of computer programs operating in a distributed system of computers, comprising the steps of:

   dividing a computer program into a plurality of independent self-contained functional modules (10), whose interactions with one another is via one or more servers (12) being interconnected in said distributed system, whereby none of said functional modules shares data with any other module, and whereby all of said functional modules maintains all necessary state data for their own operation;

   controlling the state of all of said functional modules (10) by availability management means, which monitor the performance of said functional modules, and which coordinate detection and recovery of system failures;

loading a first copy of a functional module into a first processor's (22) address space (20) and locating a second copy of said functional module into a second processor's (22') address space (20'), said two address spaces being physically separated and independent, and said second processor's address space being known only by the according first processor;

said first processor executing said first functional module to send application dependent state data to said second processor where it is received by said second functional module executing on said second processor;

said first processor executing said first module, maintaining a normal application processing state and said second processor executing said second module, maintaining a secondary state knowledge sufficient to enable it to become a primary functioning module;

said first processor executing said first module maintaining open sessions with a a plurality of said servers connected therewith and said second processor of executing said second module maintaining a plurality of open sessions with all of said servers in the network;

said second functional module, in response to a stimulus requiring it to assume the role of said first functional module, checking that its current state is consistent with the current state of said first functional module, followed by said second module then communicating with said servers in said network to establish synchronization with the state of said servers, terminating said first functional module and updating said servers with the new address space

all clients and servers connected in said network responding to said second module assuming the role of said first module, by directing all new or queued service requests to said second module instead of to said first module;

whereby said second module assumes the role of said first module in performing primary address space (20) operations.

2. The method of claim 1 wherein said first module and said second module communicate synchronously.

3. The method of claim 1 wherein said first module and said second module communicate asynchronously.

4. The method of claim 1 wherein said second module retains a complete copy of the state of said first module.

5. The method of claim 1 wherein said second module retains a trailing copy of the state of said first module.

6. The method of claim 1 wherein said second module retains knowledge of the stimuli currently in process by said first module.

Patentansprüche

1. Ein Verfahren zur Erhöhung der Betriebsverfügbarkeit eines Systems von Computerprogrammen in einem verteilten Computersystem, das folgende Schritte umfaßt:

die Unterteilung eines Computerprogramms in zahlreiche unabhängige funktionale Module (10), die über einen oder mehrere Server (12), die im verteilten System miteinander verbunden sind, untereinander in Wechselwirkung treten, wobei keines der funktionalen Module Daten mit einem anderen Modul teilt, und wobei alle funktionalen Module sämtliche notwendigen Zustandsdaten für ihren eigenen Betrieb speichern;

die Steuerung des Zustands aller funktionalen Module (10) durch Verfügbarkeitsmanagementmittel, die die Leistung der funktionalen Module überwachen und die Erfassung und Behebung von Systemfehlern koordinieren;

das Laden einer ersten Kopie eines funktionalen Moduls in einen Adreßraum (20) eines ersten Prozessors (22) und das Versetzen einer zweiten Kopie des funktionalen Moduls in einen Adreßraum (20') eines zweiten Prozessors (22'), wobei beide Adreßräume physisch getrennt und unabhängig voneinander sind, und der Adreßraum des zweiten Prozessors nur dem entsprechenden ersten Prozessor bekannt ist;

wobei der erste Prozessor das erste funktionale Modul ausführt, um anwendungsabhängige Zustandsdaten zum zweiten Prozessor zu senden, wo diese vom zweiten funktionalen Modul empfangen werden, das auf dem zweiten Prozessor ausgeführt wird;

wobei der erste Prozessor das erste Modul ausführt und

einen normalen Anwendungsverarbeitungszustand hält, und der zweite Prozessor das zweite Modul ausführt und ausreichende Daten über den zweiten Zustand hält, die es ihm ermöglichen, zum ersten funktionalen Modul zu werden;

wobei der erste Prozessor, der das erste Modul ausführt, offene Sitzungen mit zahlreichen der Server hat, die mit ihm verbunden sind, und der zweite Prozessor, der das zweite Modul ausführt, zahlreiche offene Sitzungen mit allen Servern im Netzwerk hat;

wobei das zweite funktionale Modul als Reaktion auf einen Stimulus, der es veranlaßt, die Rolle des ersten funktionalen Moduls zu übernehmen, prüft, daß sein aktueller Zustand konsistent mit dem aktuellen Zustand des ersten funktionalen Moduls ist, worauf das zweite Modul mit allen Servem im Netzwerk kommuniziert, um eine Synchronisierung mit dem Zustand der Server herzustellen, wonach das erste funktionale Modul aufgegeben und die Server mit dem neuen Adreßraum aktualisiert werden;

wobei alle im Netzwerk angeschlossenen Clients und Server auf das zweite Modul reagieren, das die Rolle des ersten Moduls übernommen hat, indem alle neuen oder in Warteschlangen befindlichen Dienstanforderungen zum zweiten Modul anstatt zum ersten Modul geleitet werden;

wobei das zweite Modul die Rolle des ersten Moduls übernimmt, indem es Operationen eines primären Adreßraums (20) durchführt.

2. Das Verfahren nach Anspruch 1, bei dem das erste Modul und das zweite Modul synchron kommunizieren.

3. Das Verfahren nach Anspruch 1, bei dem das erste Modul und das zweite Modul asynchron kommunizieren.

4. Das Verfahren nach Anspruch 1, bei dem das zweite Modul eine vollständige Kopie des Zustands des ersten Moduls hält.

5. Das Verfahren nach Anspruch 1, bei dem das zweite Modul eine Nachfolgekopie des Zustands des ersten Moduls hält.

6. Das Verfahren nach Anspruch 1, bei dem das zweite Modul die Stimuli kennt, die gerade vom ersten Modul verarbeitet werden.

Revendications

1. Un procédé pour augmenter la disponibilité opérationnelle d'un système de programmes d'ordinateur fonctionnant dans un système d'ordinateur distribué, comprenant les étapes consistant à :

diviser un programme d'ordinateur en une pluralité de modules fonctionnels (10) auto-contenus, indépendants, dont les interactions mutuelles se font via un ou plusieurs serveurs (12) qui sont interconnectés dans ledit système distribué, dans lequel aucun desdits modules fonctionnels ne partage des données avec un quelconque autre module, et dans lequel la totalité desdits modules fonctionnels conservent toutes les données d'état nécessaires pour leur propre fonctionnement;

commander l'état de la totalité desdits modules fonctionnels (10) à l'aide de moyens de gestion de la disponibilité, qui surveillent la performance desdits modules fonctionnelle et qui coordonnent la réflexion et la récupération des erreurs système;

charger une première copie d'un module fonctionnel dans un espace d'adresse (20) de premier processeur (22) et placer une deuxième copie dudit module fonctionnel dans un espace d'adresse (20') de deuxième processeur (22'), lesdites deux espaces d'adresse étant physiquement séparés et indépendants et ledit deuxième espace d'adresse de deuxième processeur étant connu seulement par le premier processeur correspondant;

ledit premier processeur exécutant ledit premier module fonctionnel afin d'envoyer des données d'état dépendantes de l'application audit deuxième processeur, dans lequel elles sont reçues par ledit deuxième module fonctionnel en cours d'exécution sur ledit deuxième processeur;

ledit premier processeur, exécutant ledit premier module maintenant, un état de traitement d'application normal et ledit deuxième processeur, exécutant ledit deuxième module, maintenant une connaissance de l'état secondaire suffisante, pour lui permettre de devenir un module à fonctionnement primaire;

ledit premier processeur exécutant ledit premier module maintenant des sessions ouvertes avec une pluralité desdits serveurs lui étant connectée et ledit deuxième processeur exécutant ledit deuxième module maintenant une pluralité de sessions ouvertes avec la totalité desdits serveurs dans le réseau;

ledit deuxième module fonctionnel, en réponse à un stimulus lui demandant de prendre le rôle dudit premier module fonctionnel, contrôlant que son état actuel est cohérent avec l'état actuel dudit premier module fonctionnel, suivi par le fait que ledit deuxième module communique ensuite avec lesdits serveurs dans ledit réseau, pour établir la synchronisation avec l'état desdits serveurs, en finissant par ledit premier module fonctionnel et la mise à jour desdits serveurs par le nouvel espace d'adresse;

la totalité des clients et serveurs qui sont connectés dans ledit réseau répondant audit deuxième module en prenant le rôle dudit premier module, en dirigeant la totalité des requêtes de service, qu'elles soient nouvelles ou mises en file d'attente, vers ledit deuxième module au lieu de les diriger vers ledit premier module;

de manière que ledit deuxième module prenne le rôle dudit premier module en effectuant les opérations d'espace d'adresse primaires (20).

2. Le procédé selon la revendication 1, dans lequel ledit premier module et ledit deuxième module communiquent de façon synchrone.

3. Le procédé selon la revendication 1, dans lequel ledit premier module et ledit deuxième module dans lequel ledit premier module et ledit deuxième module communiquent de façon asynchrone.

4. Le procédé selon la revendication 1, dans lequel ledit deuxième module conserve une copie complète de l'état dudit premier module.

5. Le procédé selon la revendication 1, dans lequel ledit deuxième module conserve une copie récapitulative de l'état dudit premier module.

6. Le procédé selon la revendication 1, dans lequel ledit deuxième module conserve la connaissance des stimuli actuellement mis en oeuvre par ledit premier module.

FIG. 1

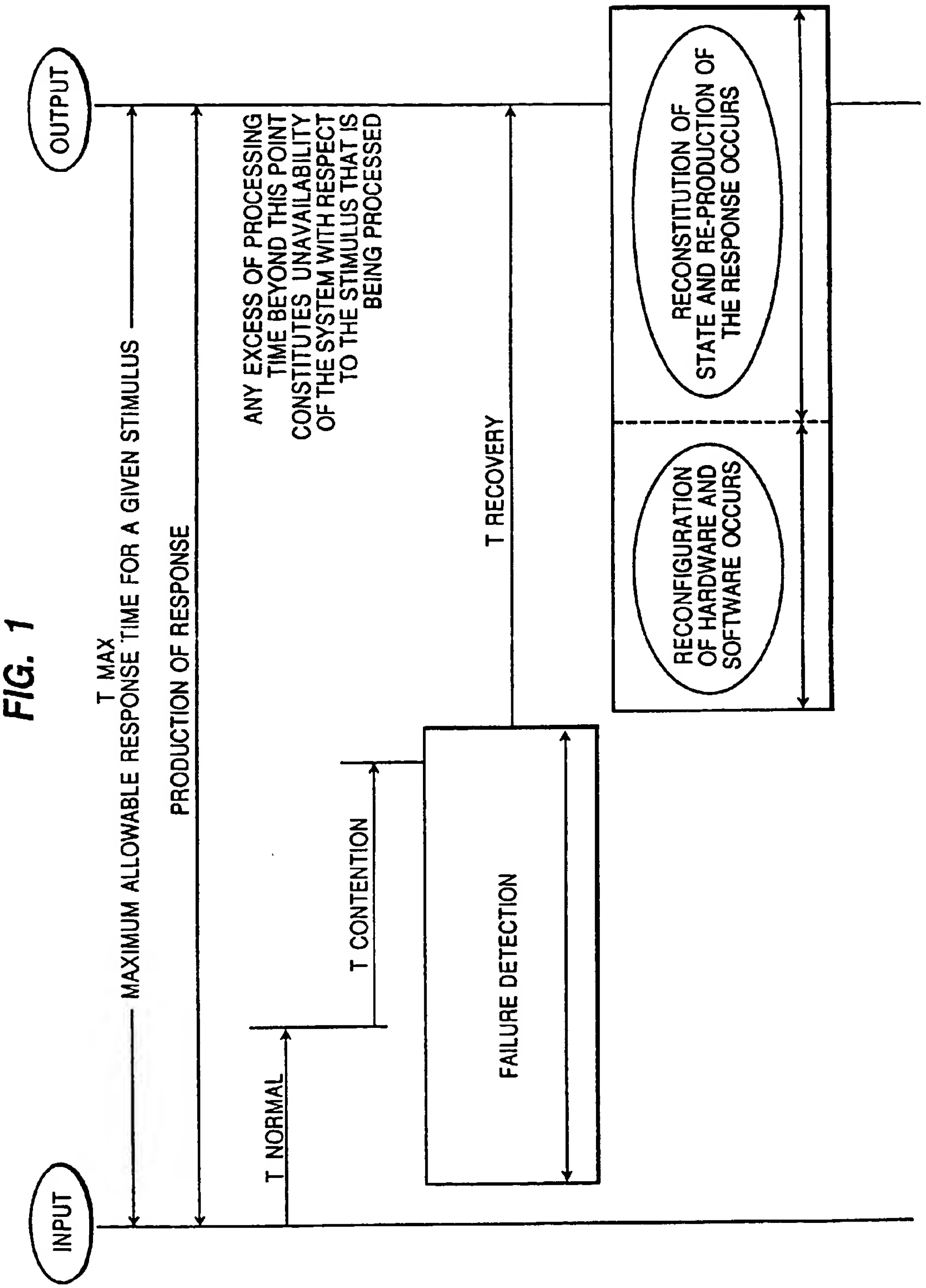

INPUT
OUTPUT
T MAX
MAXIMUM ALLOWABLE RESPONSE TIME FOR A GIVEN STIMULUS
PRODUCTION OF RESPONSE
ANY EXCESS OF PROCESSING TIME BEYOND THIS POINT CONSTITUTES UNAVAILABILITY OF THE SYSTEM WITH RESPECT TO THE STIMULUS THAT IS BEING PROCESSED
T NORMAL
T CONTENTION
T RECOVERY
FAILURE DETECTION
RECONFIGURATION OF HARDWARE AND SOFTWARE OCCURS
RECONSTITUTION OF STATE AND RE-PRODUCTION OF THE RESPONSE OCCURS

FIG.2A

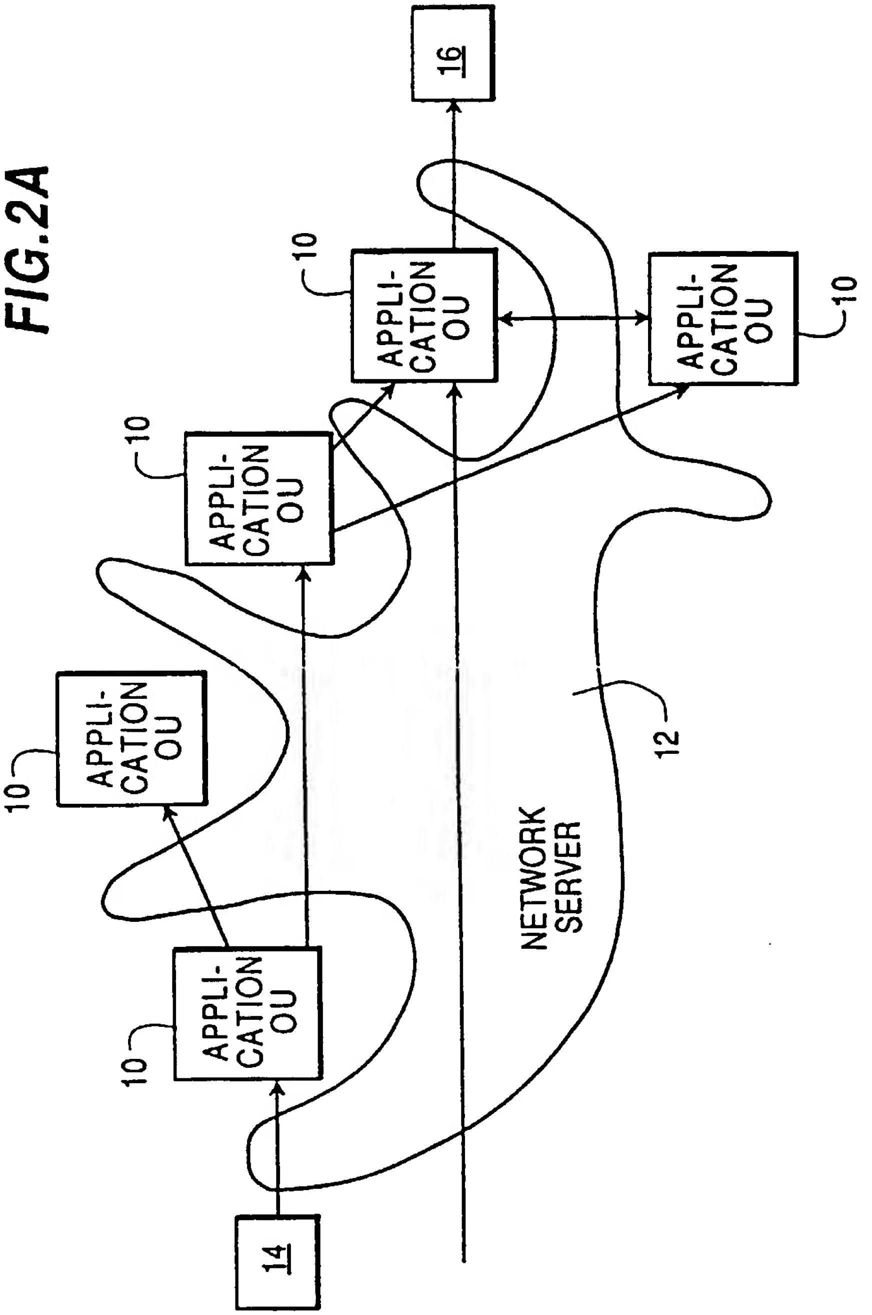

14
10
APPLI-CATION OU
10
APPLI-CATION OU
10
APPLI-CATION OU
10
APPLI-CATION OU
10
APPLI-CATION OU
16
NETWORK SERVER
12

FIG.2B

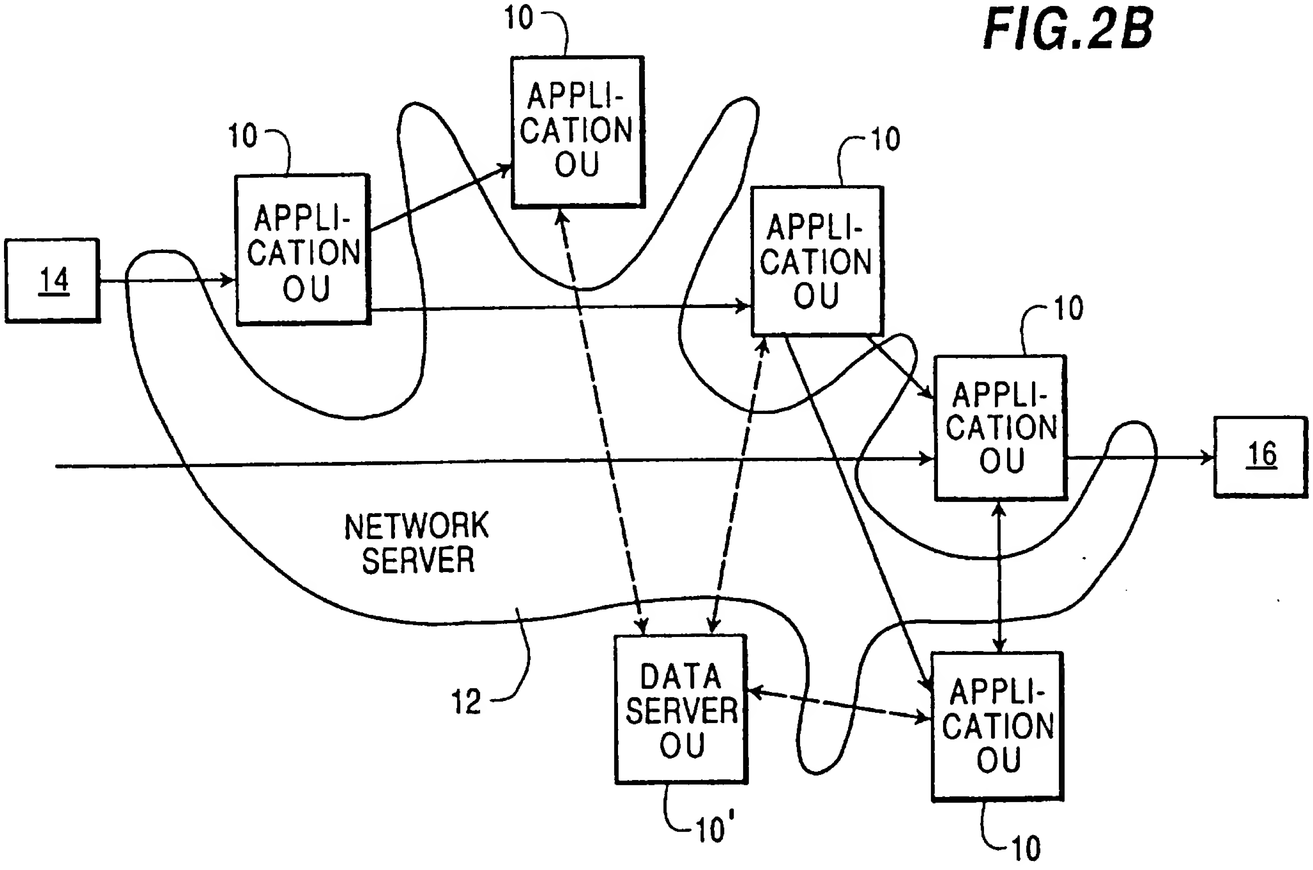
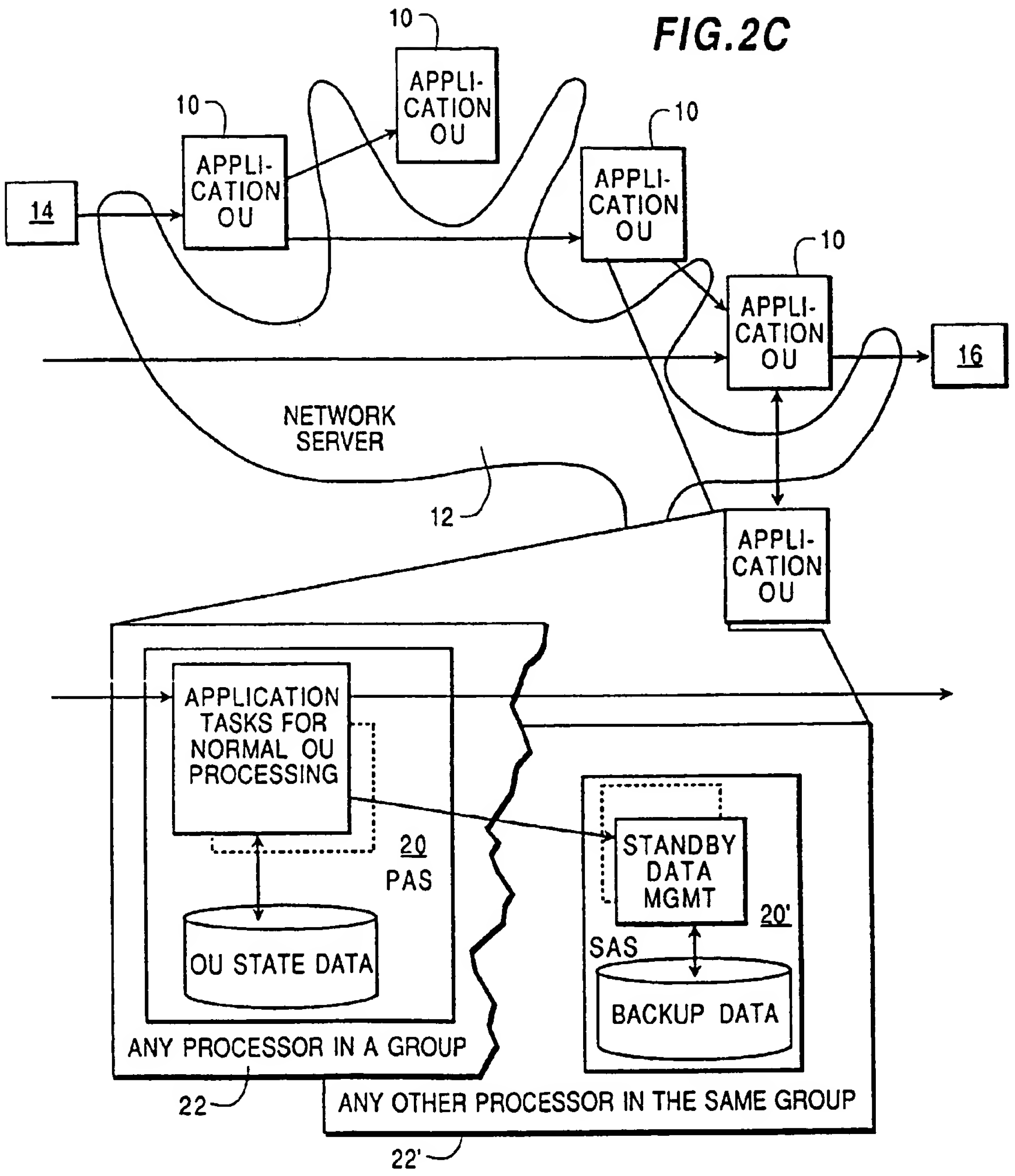

10
APPLI-
CATION
OU
10
APPLI-
CATION
OU
14
10
APPLI-
CATION
OU
10
APPLI-
CATION
OU
16
NETWORK
SERVER
12
DATA
SERVER
OU
10'
APPLI-
CATION
OU
10

FIG.2C


10
APPLI-CATION OU
10
APPLI-CATION OU
10
APPLI-CATION OU
10
APPLI-CATION OU
14
16
NETWORK SERVER
12
APPLI-CATION OU
APPLICATION TASKS FOR NORMAL OU PROCESSING
20
PAS
OU STATE DATA
ANY PROCESSOR IN A GROUP
22
STANDBY DATA MGMT
20'
SAS
BACKUP DATA
ANY OTHER PROCESSOR IN THE SAME GROUP
22'

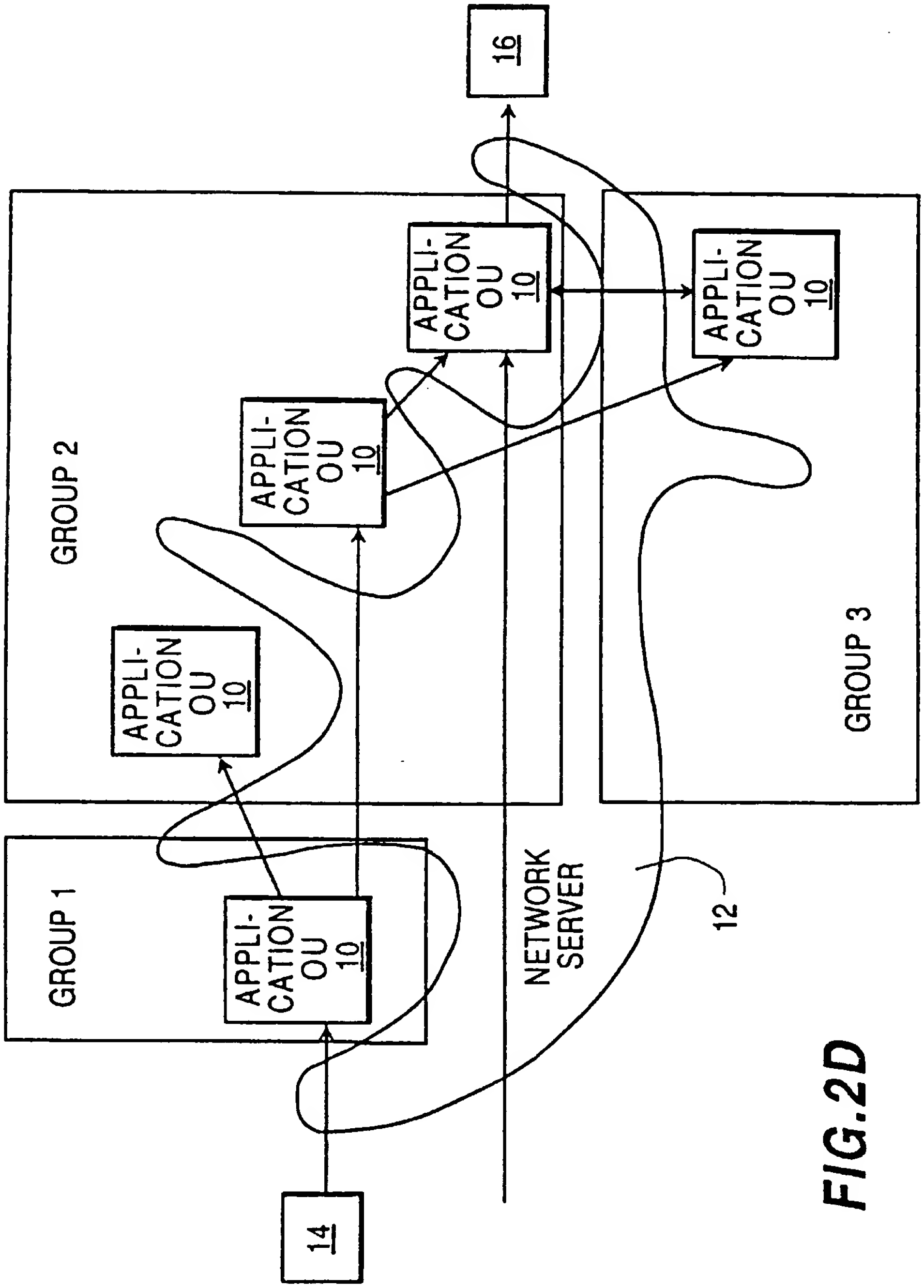

GROUP 1
GROUP 2
GROUP 3
APPLI-CATION OU 10
APPLI-CATION OU 10
APPLI-CATION OU 10
APPLI-CATION OU 10
APPLI-CATION OU 10
14
16
NETWORK SERVER
12

FIG.2D

FIG. 3A

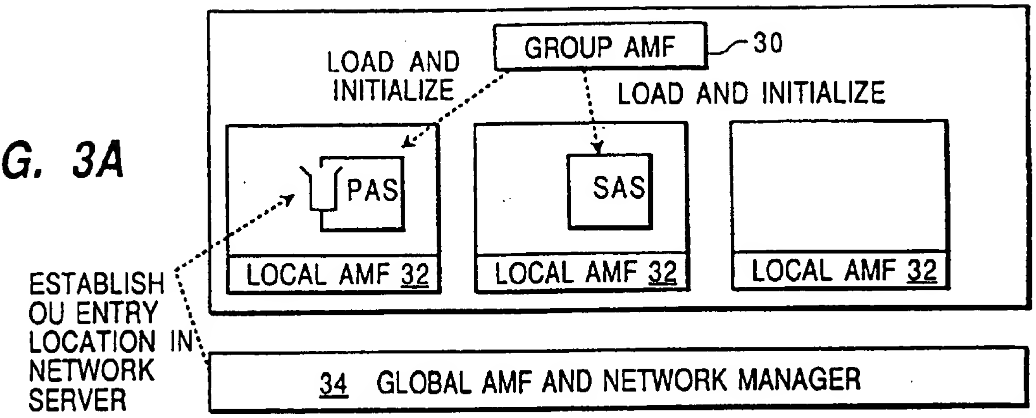


FIG. 3B

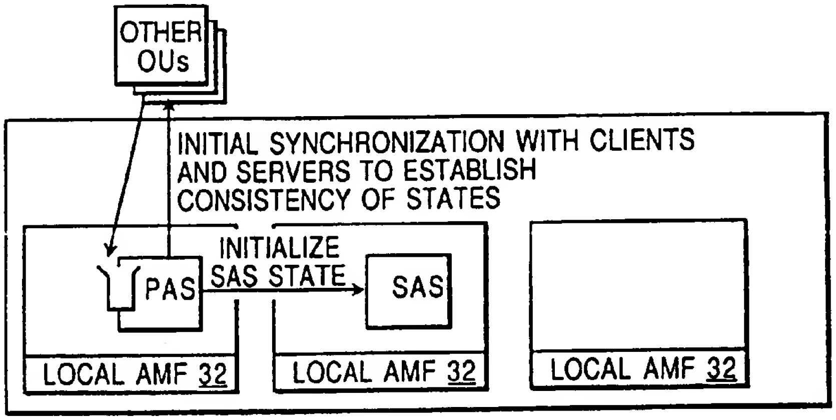

OTHER OUs
INITIAL SYNCHRONIZATION WITH CLIENTS AND SERVERS TO ESTABLISH CONSISTENCY OF STATES
INITIALIZE SAS STATE
PAS
SAS
LOCAL AMF 32
LOCAL AMF 32
LOCAL AMF 32

FIG. 3C

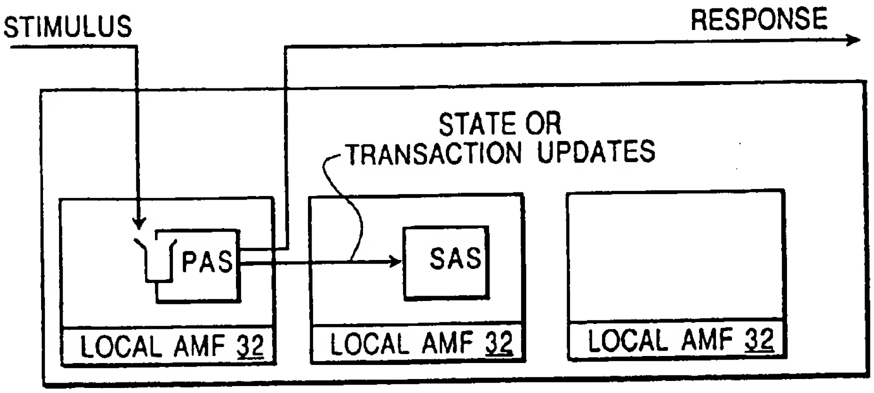


FIG. 3D

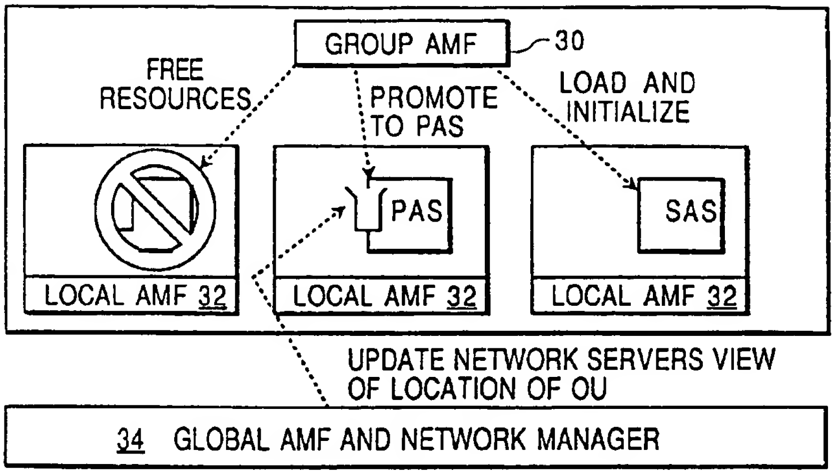


FIG. 3E

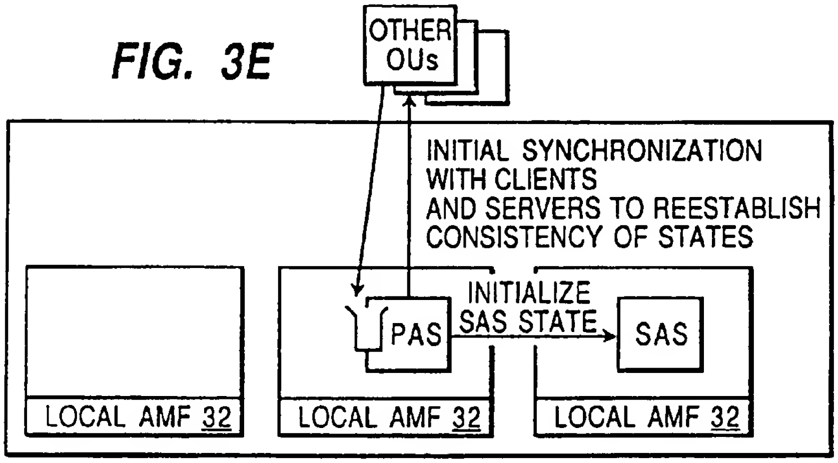
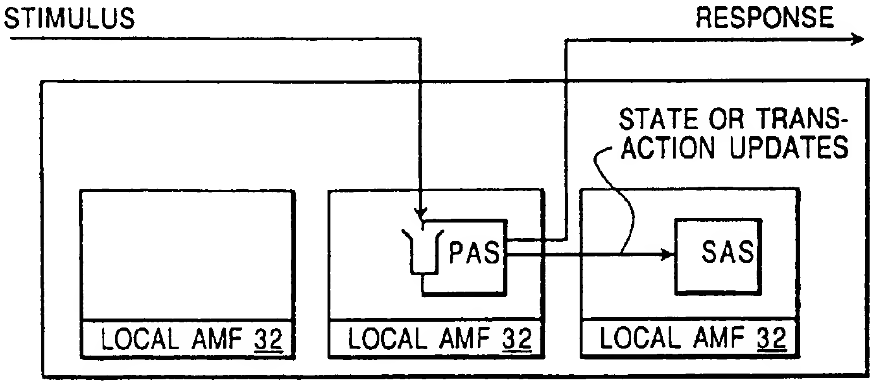

OTHER OUs
INITIAL SYNCHRONIZATION WITH CLIENTS AND SERVERS TO REESTABLISH CONSISTENCY OF STATES
INITIALIZE SAS STATE
PAS
SAS
LOCAL AMF 32
LOCAL AMF 32
LOCAL AMF 32

FIG. 3F


STIMULUS
RESPONSE
STATE OR TRANSACTION UPDATES
PAS
SAS
LOCAL AMF 32
LOCAL AMF 32
LOCAL AMF 32